RELATED APPLICATIONS

[0001] This application is related to U.S. application Ser. No. 09/750,293, (Attorney Docket: 3175-51A), entitled "Dual Mode Data Compression Technique" and filed concurrently herewith on Dec. 29, 2000.

TECHNICAL FIELD

[0002] The present application relates generally to data compression and more particularly to an enhanced data compression technique. This technique is particularly suitable for use in the graphic arts for compressing large images.

BACKGROUND OF THE INVENTION

[0003] In the graphic arts there is a tendency to have extremely large, one-bit-per-sample images approaching or even exceeding 2 gigabytes of data. The need to compress such data has been well known for many years.

[0004] One proposed technique for compressing such data is commonly referred to as a PackBits, hereinafter "PB," compression technique. The PB compression technique produces either a string of characters preceded with a count and a repeat character code or, alternatively, a single byte pattern preceded with a count. The PB compression technique is capable of processing data very quickly. This technique also provides satisfactory results when the data is a string of solid black or solid white, digitally represented in binary form by a repeating string of 1s or 0s respectively. Accordingly, the PB technique provides reasonably satisfactory results for non-color image data.

[0005] An exemplary PackBits representation of a stream of sequential input data, as it would appear entering a processor prior to encoding, might include the string of characters "abc0000000000". Using the PB technique, the processor would first determine if a first

character "a" and a second character "b" are the same character. In some proposed PB techniques, the processor might scan ahead to consider subsequent characters when determining if a stream contains the same repeated character. In the present example, the comparison that determines if "a" and "b" are the same character returns a negative result. The processor then proceeds to encode the input data as a literal string with a length. Next the processor determines if the second character "b" and the third character "c" are the same character. Since this determination is also negative, the processor will proceed to encode the three characters of the input data as a literal string with a length. The processor then determines if the third character "c" and the fourth character "0" are the same character. Since this determination is also negative, the processor will proceed to encode the four characters of the input data as a literal string with a length. The processor continues and determines if the fourth character "0" and the fifth character "0"are the same character. Since this determination is positive, the processor continues by repeatedly determining if the immediately subsequent characters in the sequence are also the same character until it makes a negative determination. The processor thereby determines the repeat count for the character "0". Based on the initial positive determination, the processor also proceeds to encode the first three characters of the input data sequence, i.e. "a,""b" and "c," as a literal string with a length and the following 10 characters of the input data sequence, i.e. the "0" . . . "0," as a repeat character with a count.

[0006] Accordingly, the processor generates encoded output data forming a 2-byte sequence including the strings of characters "82abc" and "090". In the output data, the "8" serves as a header indicating that the total length of the sequence is 8 bits and that a literal string follows. The "2" indicates that the length of the literal string is three characters, i.e. characters "a," "b," and "c". The first "0" indicates that a repeat character follows, and the "9" indicates that the repeat character, the second "0," is repeated 10 times. Using one-off numbers such as the "2" to indicate a literal string of 3 characters, and the "9" to indicate that a repeat character is repeated 10 times. Using one-off numbers such as the "2" to indicate a literal string of 3 characters, and the "9" to indicate that a repeat character is repeated 10 times, is efficient because 129 bytes can be packed using number up to 128.

[0007] To decode the encoded sequence "82abc090," the receiving processor first reads the header "8," which is the highest order bit. From the header, the processor determines that a literal string follows. The processor then extracts the string length, "2," and reads the next three characters "a," "b" and "c". At this stage, the output string is "abc" and the remaining input string is "090." The processor then reads the first "0" which indicates that a repeat character follows. The processor continues by extracting the repeat count, "9," and then reads the next character "0," the character to be repeated 10 times. The resulting decoded string is "abc0000000000," is the string originally presented prior to encoding.

[0008] As should be clear from the above, the PB technique processes only one character at a time. Accordingly, PB is incapable of compressing strings of repeating multi-byte patterns of characters. The PB technique also has a relatively limited compression rate, generally no more than 64 to 1. Thus, the PB compression technique provides unsatisfactory results when used to compress color image data, which typically contains repeating multi-byte patterns of characters instead of repeating single-byte 0s and 1s.

[0009] Another proposed technique for compressing image data is the Lempel-Ziv-Welch, hereinafter "LZW," compression technique. Using the LZW compression technique, variable length strings of byte based data can be processed. The LZW compression technique processes the data somewhat slower than the PB compression technique, but provides satisfactory results on data representing both color images and black-and-white images. However, since these techniques are based on single bytes of data, such techniques are incapable of compressing data on an arbitrary pixel or bit boundary basis. Additionally, though LZW is capable of providing a higher compression rate than PB, LZW's compression rate is still somewhat limited.

[0010] Using the LZW technique, the encoding and decoding processors must coordinate the transmission and receipt of codes. The LZW technique uses a compression dictionary containing some limited number of compression codes defined during the processing of the input data. The

characters in the input string are read on a character by character basis to determine if a sub-string of characters match a compression code defined during the processing of prior characters in the input string. If a pattern match is found, the matching sub-string of characters is encoded with the applicable compression code. If a sub-string of characters does not match a pre-existing code, a new code corresponding to the sub-string is added to the dictionary. Sub-strings are initially defined by codes having 9 bits, but the number of bits may be increased up to 12 bits to add new codes. Once the 12-bit limit is exceeded, the dictionary is reset and subsequent codes are again defined initially with 9 bits. In conventional implementations of the LZW technique, two codes are predefined, i.e. defined prior to initiating processing of the input string. In the present example these codes are the code 100, representing a "reset," and the code 101, representing an "end." In the present example, the codes 102, 103, 104, etc. represent strings of new patterns that are identified during the processing of the input data.

[0011] An exemplary LZW representation of a stream of sequential input data, as it would appear entering a processor prior to encoding, might include the string of characters "abc0lc0l". Using the LZW technique, the encoding processor first reads the "a" in the sequence and the "b" immediately thereafter. The processor then determines if a code exists for the character sequence "ab". Since, in this example, no such code exists at this point in the processing, a new code 103 is generated to represent the new pattern string "ab" and is added to the existing code dictionary. The processor continues by reading the "c" immediately following the "b" in the sequence. The processor determines if a code exists for the character sequence "bc". Since, in this example, no such code exists at this point in the processing, a new code 104 is generated to represent the new pattern string "bc" and the code is added to the code dictionary.

[0012] The processor continues and reads the "0" immediately following the "c" in the sequence. The processor determines if a code exists for the character sequence "c0". Since, in this example, no such code exists at this point in the processing, a new code 105 is generated to represent the new pattern string "c0" and the code is added to the code dictionary. The processor continues and reads the "1" immediately following the "0" in the sequence. The processor

determines if a code exists for the character sequence "01". Since, in this example, no such code exist at this point in the processing, a new code 106 is generated to represent the new pattern string "01"and the code is added to the dictionary. The processor proceeds and reads the "c" immediately following the "1" in the sequence. The processor determines if a code exists for the character sequence "1c". Since, in this example, no such code exists at this point in the processing, a new code 107 is generated to represent the new pattern string "1c" and the new code is added to the dictionary.

[0013] The processor proceeds by reading the "0" immediately following the second "c" in the sequence. The processor determines if a code exists for the character sequence "c0". In this example, such a code, i.e. code 105, does exist. The processor therefore, determines if a longer pattern match can be made, and reads the "1" immediately following the second "c0" in the sequence. The processor determines if a code exists for the character sequence "c01". Since, in this example, such a code does not exist, a new code 108 is generated to represent the new pattern string "c01," which can also be represented as "1051". The processor ultimately generates encoded output data that forms the sequence of characters: "100abc01c1051". The sequence, broken down into symbols represents: a reset (100); a literal string (abc01c); a previously found pattern (105); and a literal (1).

[0014] Using the LZW technique, the encoding processor builds a tree of codes generated using other codes. This is a primary reason why the LZW technique provides satisfactory results even though processing is performed on a byte by byte basis to find repeating byte patterns. That is, the downstream encoding builds on the upstream encoding. However, using the LZW technique, the encoding processor can take significant processing time to encode large sequences. For example, if there is a large occurrence of adjacent 0s or 1s, a significant period of time will be required by the processor to encode the sequence.

[0015] The decoding processor builds a similar tree from the codes received from the encoding processor. Basically, the decoding processor performs the reciprocal of the encoding process to decode the encoded sequence characters "100abc011051".

[0016] In summary, the PB compression technique is deficient in that it addresses only single byte repeats and is limited to a 64 to 1 compression rate. Therefore, it is not suitable for color images. On the other hand, while the LZW compression technique addresses multi-byte repeats and has a compression rate of perhaps 500 to 1, it requires significant processing time to build the codes that are required to obtain good compression. Hence, although the LZW technique may be suitable for encoding relatively small amounts of data, when encoding gigabytes of data, such as an 80 inch x 50 inch image having 2400 dots per inch, the processing time and/or resources necessary to encode data make using the LZW technique alone impractical.

[0017] Accordingly, the need exists for a technique which can quickly compress large amounts of image data, offer a still higher compression rate than previously proposed techniques, and provide satisfactory results when used to compress either color or non-color image data.

OBJECTIVE SUMMARY OF THE INVENTION

[0018] It is an object of the present the invention provides a technique for quickly compressing large amounts of image data.

[0019] It is a further object of the present the invention provides a technique which facilitates high compression rates for both color and non-color image data.

[0020] The invention provides a technique that gives satisfactory results when used to compress both color and non-color image data.

SUMMARY DISCLOSURE OF THE INVENTION

[0021] In accordance with the invention, an encoder includes a memory configured to store a predefined compression code for one of white image data and black image data. The memory, which may take the form of a hard, floppy, compact or optical disk, RAM, ROM, or some other type of memory, stores a predefined compression code corresponding to white image data or black image data. The term predefined is used herein to mean that the compression code(s) are defined irrespective of the input data which will be compressed using these codes. In practice, the compression code(s) will typically be defined prior receipt of the input data. The encoder also includes a processor configured to convert a first sequence of characters representing an image into a second sequence of characters including the stored predefined compression code.

[0022] In a preferred implementation, the processor receives a first sequence of characters representing an image. A first character in the sequence is read. The processor determines if the read character represents the white or black image data. If so, one or more characters occurring immediately subsequent to the first character in the sequence of characters are read. The processor determines if the one or more characters match the read first character and, if so, generates a second sequence of characters, including the stored predefined compression code, representing the matching one or more characters.

[0023] Preferably, the memory stores two predefined compression codes corresponding respectively to white image data and black image data. In such an implementation, the processor may receive a third sequence of characters representing the image, read a first character in the third sequence, and determine if the read character in the third sequence represents white or black image data. If so, the processor reads one or more characters occurring immediately subsequent to the first character in the third sequence of characters, and determines if these read characters match the first character in the third sequence. If so, the processor generates a fourth sequence of characters, including the applicable stored predefined compression code, representing the matching characters in the third sequence.

[0024] According to other aspects of the invention, the encoder memory may store a threshold value. Preferably, the threshold valve corresponds to a desired minimum compression rate. If such a threshold is provided, the processor determines if a value corresponding to the number of characters in the matching one or more characters is equal to or greater than the threshold value. Only if the threshold is met or exceeded is the second sequence of characters, including the stored predefined compression code, generated.

[0025] According to still other aspects of the invention, the processor generates the second sequence of characters so as to also include a value corresponding to the number of characters in the matching one or more characters. This value is commonly referred to as a repeat count value.

[0026] The second sequence of characters may be limited to a predefined bit length, such as 9, 10, 11 or 12 bits. In such a case, the second sequence of characters is preferably capable of including a continuation code, if appropriate. For example, if the number of repeating characters is very large and therefore cannot be entirely represented in a single code having a bit length within the predefined limit, a second code will be required to complete the encoding. Therefore, the processor generates a third sequence of characters, excluding the stored predefined compression code, to represent those of the matching one or more characters not represented by the second sequence of characters. It will be recognized that more than two sequences may be necessary to fully represent all of the matching one or more characters if the number of matching characters is extremely large. Typically the processor combines the second, third, and, if applicable, other sequences of characters to represent the matching characters in their entirety.

[0027] In another implementation of the invention, an imaging system includes a raster image processor and an imager controller. The imager controller could be a pre-press imager controller, a printer controller, a television display controller, a computer monitor controller, or some other type of image rendering device controller. The raster image processor receives a first sequence of characters representing an image and converts the first sequence of characters into a second

sequence of characters including a predefined compression code for white image data or black image data. The imager controller receives the second sequence of characters representing the image and coverts the second sequence of characters into the first sequence of characters based on the predefined compression code for the white or black image data, as applicable.

[0028] Preferably, the raster image processor stores the predefined compression code, and coverts the first sequence of characters by reading a first character in a first sequence of characters, and determining if the read character represents the white or black image data. If so, the raster image processor reads one or more characters occurring immediately subsequent to the first character in the first sequence of characters and determines if the read one or more characters match the read first character. If so, the raster image processor proceeds to generate the second sequence of characters to represent the matching one or more characters.

[0029] Advantageously, the predefined compression code is a first predefined compression code and corresponds to the white image data. The raster image processor further receives a third sequence of characters representing the image and converts the third sequence of characters into a fourth sequence of characters including a second predefined compression code corresponding to the black image data. The imager controller receives the fourth sequence of characters representing the image and converts the fourth sequence of characters into the third sequence of characters based on the second predefined compression code.

[0030] The raster image processor is also preferably configured to determine if a value corresponding to the number of characters in the first sequence of characters meets or exceeds a threshold value. Only if the corresponding value is equal to or greater than the threshold value, does the raster image processor generate the second sequence of characters, including the predefined compression code.

[0031] Typically, the raster image processor also generates the second sequence of characters so as to include a value corresponding to the number of characters in the first sequence of characters.

BRIEF DESCRIPTION OF THE DRAWINGS

[0032] FIG. 1 depicts an exemplary simplified depiction of an image processing system in accordance with a first embodiment of the present invention.

[0033] FIG. 2 depicts an exemplary simplified depiction of an image processing system in accordance with a second embodiment of the present invention.

[0034] FIG. 3 depicts an exemplary code dictionary in accordance with the second embodiment of the present invention.

[0035] FIG. 4, FIG. 5, and FIG. 6 are flowcharts of exemplary embodiments of the invention.

[0036] Additional objects, advantages, novel features of the present invention will become apparent to those skilled in the art from this disclosure, including the following detailed description, as well as by practice of the invention. While the invention is described below with reference to preferred embodiment(s), it should be understood that the invention is not limited thereto. Those of ordinary skill in the art having access to the teachings herein will recognize additional implementations, modifications, and embodiments, as well as other fields of use, which are within the scope of the invention as disclosed and claimed herein and with respect to which the invention could be of significant utility.

DETAILED DESCRIPTION OF THE INVENTION

[0037] In pre-press imaging, particularly for flats having an entire plate worth of image information, most of the data is often either solid black or solid white, digitally represented in binary form by a stream of repeating 1s or 0s, respectively. For halftone images all of the data is black and white, i.e., 1's and 0's.

[0038] FIG. 1 is a simplified, exemplary depiction of a image processing system 1000 according to the first embodiment of the present invention. The system 1000 includes a raster image processor, hereinafter "RIP," 1050, an imager controller 1100, and an imager 1150. The raster processor includes a processor 1050a and a memory 1050b for storing processing instructions and other data. The RIP 1050 receives image data and converts the image data into encoded data. The encoded data is then transmitted to the imager controller 1100, which includes a processor 1100a and a memory 1100b for storing processing instructions and other data. The imager controller 1100 generates imager control signals based on the data received from the RIP 1050. The imager controller 1100 sends the imager control signals to the imager 1150. Specifically, the control signals from the processor 1100a control the operation of the imager scanning assembly 1150a to form the image on a medium 1150c, such as, a film or plate, supported within the imager 1150. As shown, the imager includes 1150 uses a cylindrical drum 1150b to support the medium 1150c. Alternatively, the imager 1150 could use a flat bed or external drum for supporting the medium.

[0039] In a first mode of operation, which will hereafter be referred to as a flat banding "banded mode," the RIP 1050 receives an 80 inch x 50 inch color separated image having 2400 dots per inch. The image could, for example, correspond to multiple pages of a magazine. In such a case, using imposition software on a front end preprocessor (not shown), the image could be formatted such that the image printed from the imaged medium 1150c is positioned to facilitate cutting, folding, and stitching to create multiple properly printed and positioned magazine pages.

The RIP 1050 converts the entire image into multiple gigabytes of data encoded data as a single job.

[0040] However, due to processing power limitations of RIP 1050, the entire image cannot be converted into encoded data in a single operational process. Instead, the RIP 1050 slices the image into bands prior to the data being converted. The RIP processor 1050a may perform the banding of the image or a preprocessor (not shown) could also perform the banding of the image. In the preferred embodiment, the RIP processor 1050a encodes the image data in each of the bands in a separate operational process. Thus, the job of encoding all the image bands, and therefore the entire image, is completed only after the RIP 1050 performs multiple, separate operational processes. In practice, the larger the image, the smaller each image band is, with all bands preferably being equal in size. Furthermore, the larger the image, the greater the startup time required before encoding can begin. This limitation is caused by the increased pre-encoding processing for larger images. Additionally, the more objects included in the image, the more memory that is required.

[0041] In the second mode of operation, hereinafter "page assembly mode," the RIP 1050 receives, as multiple smaller images, an 80 inch x 50 inch color image having 2400 dots per inch. In this case, one of the multiple images is primarily white. This image might be a template image and include information such as registration marks, color gradients, and identification marks. The other images could, for example, be the images for pages of a magazine, each image being a separate page. Here, the RIP 1050 may be operated to encode the image data representing the template image as one job and to encode the data of the other images as another job. When both jobs are complete, the entirety of the image will be encoded.

[0042] In the page assembly mode, the image is divided into page assemblies. One of the pages is a template image that is, primarily white, and that is typically processed without being split into bands. The other pages, however, are typically sliced into bands prior to being encoded by the RIP 1050. Because the area of each of the page images is much smaller than the area of the

entire image used in the first mode of operation, fewer bands are required. As a whole, it will take less time to encode the image data representing the multiple images in the page assembly mode than the time required to encode the image data representing the entire image in the banded mode discussed above. Thus, the RIP processor 1050a encodes the image data for each of the bands, in each of the non-template pages, in a separate operational process. The job, or jobs if the template image is pre-processed, is completed only after all of the pages, which together represent the entire image, are encoded.

[0043] Although, the image discussed in the page assembly mode description may be the same image discussed in the banded mode description, encoding in the page assembly mode will typically result in a greater amount of encoded data than encoding in the banding mode. For example, the RIP 1050 may generate two gigabytes of encoded data in the banded mode, yet generate three gigabytes of encoded data for the same image in the page assembly mode. The discrepancy is due to the page assembly mode retaining more uncompressed data.

[0044] In the banded mode, the image bands may be satisfactorily converted using the PB technique. In the page assembly mode, a template image may be satisfactorily converted using a PB technique since it is primarily white or primarily black and so is made up of mainly a repeating stream of 0s and 1s, respectively. However, the PB technique will often produce unsatisfactory results when used to convert the bands of the other of the multiple images. Accordingly, in the page assembly mode, these bands are converted using an LZW technique. Thus, in the page assembly mode, different compression techniques are utilized for a single image and perhaps even in a single job.

[0045] Referring to FIG. 4, accordingly, in one embodiment of the invention, the RIP 1050 can operate in either the banded or the page assembly mode. The RIP 1050 initially scans the received image data to determine if banded mode or page assembly mode operations are appropriate. Alternatively, the image may be sliced into bands during pre-processing. (STEP 3000). If the RIP 1050 determines (STEP 3010) that banded mode is appropriate, it encodes the

image data using the PB technique (STEP 3020). If, however, the RIP 1050 determines that page assembly mode is appropriate, it uses a different technique (STEP 3030). Referring to FIG. 5, the RIP 1050 further determines if the page image data represents a template image or banded image (STEP 3050). If the page image data represents a template image, which as described is likely to be primarily black or white, the RIP 1050 uses the PB technique to encode the template image (STEP 3020). If, however, the page image data represents a banded image, the RIP 1050 uses the LZW technique to encode the banded image data (STEP 3060). The selective operation of the RIP 1050, in response to the type of image data received, facilitates a more efficient and effective processing of large images than was previously obtained in conventional RIPs.

[0046] In a second embodiment of the invention, encoding can be interrupted and a more efficient compression technique may be applied. The invention chooses to interrupt the processing if the stream contains a section of all black or all white data. As a stream of sequential data is processed prior to encoding, if the immediately preceding character, which has yet to be encoded, matches the next character in the stream, and this next character is either solid black (e.g., a stream of all 1s), or solid white (e.g., a stream of all 0s), encoding is interrupted. During the interruption, a determination is made as to whether the invention determines if one or more characters, immediately following the next character in the sequence, also match the next character.

[0047] Another embodiment of the invention will now be described with reference to FIG. 2. As shown, FIG. 2 represents a simplified, exemplary depiction of an image processing system 2000. The system 2000 comprises a raster image processor, hereinafter "RIP," 2050, an imager controller 2100, and an imager 1150. The RIP 2050 receives an image and encodes the image data. The encoded data is then transmitted to imager controller 2100, which generates imager control signals based on decoded data received from the RIP 2050. The imager 1150 in FIG. 2 is identical to the imager 1150 of FIG. 1. Specifically, the control signals from the imager controller processor 2100a control the operation of the imager scanning assembly 1150a to form

the image on a medium 1150c. The medium could be identical to the medium 1150c in FIG. 1. The medium 1150c is supported within the imager 1150 of FIG. 2. As shown, the imager 1150 includes a cylindrical drum 1150b for supporting the medium 1150c.

[0048] In this embodiment of the present invention, the RIP processor 2050a implements a compression technique, which will hereafter be referred to as the "*AGFA* technique." The *AGFA* technique can process strings of byte based data of variable length. Using the *AGFA* technique is substantially faster then the LZW compression technique for many large image applications, while still providing satisfactory results for both color images as well as those that are primarily black-and-white. Furthermore, the *AGFA* technique is not limited to single bytes of data, and is therefore capable of compressing data on an arbitrary pixel or bit boundary basis. Additionally, the *AGFA* technique is capable of providing a higher compression rate than either the PB or the LZW compression techniques implemented separately.

[0049] An exemplary representation of a stream of sequential input data as it would appear entering a RIP processor 2050a prior to encoding could, for example, include the string of characters "abc0 . . . 01c01". The string "0 . . . 0" is a large string of zeros.

[0050] Using the *AGFA* technique, the encoding and decoding processors, i.e. the RIP processor 2050a and imager controller processor 2100a, must coordinate the transmission and receipt of codes, similar to the coordination required by the LZW technique. However, as will be described below, the *AGFA* technique uses a compression dictionary containing four pre-defined compression codes. The characters in the input string are scanned to determine if a scanned sub-string of characters match certain pre-defined compression codes. If so, the matching sub-string of characters is encoded with the applicable pre-defined compression code. If a sub-string of characters does not match a pre-existing code, a new code corresponding to the sub-string is added to the dictionary.

[0051] Furthermore, the *AGFA* technique provides a look-ahead function. The look-ahead function determines if the sub-string is greater than a minimum number, preferably 6 bytes, and if the sub-string is encoded with a new code; the new code comprising any applicable pre-existing code and the length of the code field. The length of the code field is the width of the pre-existing code, with the length forming the most significant bits and serving as a continuation indicator. Any new code follows the length; the new code forming the least significant bits. Like the LZW technique, sub-strings are initially defined by codes having 9 bits, but may be increased to up to 12 bits to add new codes. Once the 12-bit limit is exceeded, the dictionary is reset and subsequent codes are again defined initially with 9 bits.

[0052] Referring to FIG. 3, in the *AGFA* technique, four codes are predefined and stored in the code dictionary 1300 in the RIP's memory 2050b as codes 1330. In the present example these codes are: the code 100, representing a sub-string of all zeroes, which corresponds to solid white; the code 101, representing a sub-string of all ones, which corresponds to solid black; the code 102, representing a reset; and the code 103, representing the end of the compressed encoded data. In the present example, codes 104, 105, 106, etc. represent sub-strings of new patterns which are generated during the processing of the input data and also stored in the RIP's memory 2050b in the code dictionary 1300. It will be recognized that because codes for the strings corresponding to white and black are partially predefined. Since predefined codes can simply be read by the RIP processor 2050a from the dictionary, reduced processing is required to generate these codes.

[0053] Using the *AGFA* technique, the RIP processor 2050a first sets a reset code 102 (read from the code dictionary 1300) and reads the "a" in the sequence and the "b" immediately thereafter. The RIP processor 2050a then determines from the code dictionary 1300, if a code exists for the character sequence "ab". Since, in this example, no such code exists at this point in the processing, a new code 105 is generated to represent the new pattern string "ab" and the new code is stored in the code dictionary 1300 in memory 2050b. The RIP processor 2050a continues and reads the "c" immediately following the "b" in the sequence. The RIP processor 2050a determines if a code exists for the character sequence "bc". Since, in this example, no

such code exist at this point in the processing, a new code 106 is generated to represent the new pattern string "bc" and the new code is stored in the dictionary 1300.

[0054] Also referring to FIG. 6, the RIP 2050 continues and reads 6000 the "0" immediately following the "c" in the sequence. The RIP processor 2050a determines if a code exists for the character sequence "c0". Since, in this example, no such code exists at this point in the processing, a new code 107 is generated to represent the new pattern string "c0" and the new code is stored in the dictionary 1300. Also, because the "0" is recognized as a special character 6010, the RIP processor 2050a automatically scans ahead 6020 to read the next character to determine if it matches 6030 the initial "0". If the next character is not a matching "0," the scanning ahead is immediately discontinued and the RIP processor 2050a proceeds with normal processing 6040. If the next character 6050 is a matching "0," 6060 the scanning ahead continues on, character by character 6050-6060, until no matching "0" is found. At that point 6070, the scanning ahead is discontinued and normal processing continues.

[0055] In this exemplary application of the *AGFA* technique, the RIP processor 2050a scans ahead and counts the number of repeated "0" or "1" bytes in the sequence. Preferably, a compression threshold is pre-established and stored in the RIP memory 2050b. For example, the threshold might correspond to a 4 to 1 compression rate. If such a threshold is utilized 6080, and the number of repeated "0" or "1" bytes counted is less than the number required to meet or exceed the threshold, e.g. if the sequence consists of only one or two zeros or ones, then a new code would be established for the sequence in the normal manner 6040. Only if the number of repeated "0" or "1" bytes counted meets or exceeds the threshold is the sequence encoded 6070 using the applicable pre-defined code 100 or 101.

[0056] Assuming in the present example that the number of "0" bytes counted by the RIP processor 2050a meets or exceeds the threshold, the bits in the "count position" represent a repeat count. Either 9, 10, 11, or 12 bits can be used to code the repeat count. However, if the count is so great that more than 11 bits would be required for the encoding, a continue code

which may be generated by the processor 2050a or retrieved from memory 2050b, is inserted as the least significant bit in the output code. This continue code enables the output codes representing the entire sequence of zeros or ones to be strung together. Therefore, no matter how long the sequence is, the low or least significant bit of each output code within the string of output codes would represent either an end code or a continuation of the coding. Hence, 1 bit is sacrificed for the end/continuation bit leaving 8, 9, 10, or 11 bits for the repeat count.

[0057] Accordingly, in the present example, the output code for the repeat count of "0" characters would be formed using the code "100" to indicate that this is a sequence of "0" characters, followed by "102" representing a first portion of the repeat count, and "001" indicating that the output codes for the repeat count continues. Thus, the first code in the string of repeat count output codes would be "100102001". The second code in the string of repeat count output codes could be "102001," with the "102" representing a second portion of the repeat count, and "001" indicating that the output codes for the repeat count continues. The last code in the string of repeat count output codes could be "0201". The high bit of the last output code "0201" indicates that this is the end of the repeat count information in this field.

[0058] Using the repeat count multiple times, the strung-together codes for the entire repeat count would, in the above example be "100102001102001020 1". Thus, the strung together multiple bytes of output codes provide a full representation of the repeat count. In practice, five output codes may be used to represent up to four billion characters. Notwithstanding the number of bits in the output codes, the high bit is used to represent the count. Accordingly, whatever output code size is used, full advantage is taken of all available bits for the repeat count.

[0059] Conventional LZW techniques lack the ability to scan ahead. Conventional PB techniques, on the other hand, scan ahead to locate matches with whatever character has been read and must fully generate the match coding for each matching sequence. In contrast to both, the present invention scans ahead to locate matches with only selective characters, preferably only white and black, respectively represented herein by "0" and "1". Furthermore, the

invention can use a predefined code for each of the selected characters, e.g. white and black. Hence, the coding for each matching sequence need only be partially generated since predefined codes, e.g. codes 100 or 101, are pre-generated and need only be read from the code dictionary 1300. Accordingly, the present invention is capable of providing superior encoding of large images using less computing resources and computing time.

[0060] As noted above, once the RIP processor 2050a determines it is at the last "0" in the sequence, i.e. by determining from scanning ahead that the next character does not match a "0," the scanning ahead is discontinued and normal processing resumes. Thus, the RIP processor 2050a continues by reading the "1" immediately following the last "0" in the sequence. The processor 2050a determines if a code exists for the character sequence "01". Since, in this example, no such code exist at this point in the processing, a new code 108 is generated to represent the new pattern string "0l" and the new code is added to the dictionary 1300. The processor 2050a proceeds by reading the "c" immediately following the "1" in the sequence. The processor determines if a code exists for the character sequence "1c". Since, in this example, no such code exists at this point in the processing, a new code 109 is generated to represent the new pattern string "1c" and the new code is added to the dictionary 1300.

[0061] The processor 2050a then proceeds by reading the "0" immediately following the second "c" in the sequence. The processor 2050a determines if a code exists for the character sequence "c0". In this example, such a code, i.e. code 107, does exist. The RIP processor 2050a then scans ahead to determine if another "0" immediately follows this occurrence of "c0". Since, in this case the next character is not a "0," the scanning ahead is discontinued and normal processing resumes.

[0062] The processor 2050a proceeds by reading the "1" immediately following the second "c0" in the sequence. The processor 2050a determines if a code exists for the character sequence "c01". Since, in this example, such a code does not exist, a new code 110 is generated to represent the new pattern string "c01" and the new code is added to the code dictionary 1300.

Because the "1" is the last character, the combination of the last generated code and the last character can be represented as "1071". The RIP processor 2050a also scans ahead to determine if another "1" immediately follows this occurrence of "c01". Since, in this case the next character is not a "1," the scanning ahead is discontinued. Normal processing would resume if further characters remained to be encoded. However, since "c01" and 1 are the final characters, encoding ends.

[0063] The processor 2050a ultimately generates encoded output data of the form "102abc100102001102001020110711103". The sequence includes the encoded string and an end code, code 103.

[0064] Similar to the LZW technique, in the *AGFA* technique the RIP processor 2050a builds a tree of numerous codes using a combination of pre-defined and generated codes. The *AGFA* technique is thereby capable of providing satisfactory results even though the processing is performed on a byte by byte basis to find repeating bytes. Compared to the LZW technique, the *AGFA* technique requires substantially reduced processing time and resources to encode large sequences because it uses special pre-defined codes. The decoding processor, i.e. the imager controller processor 2100a builds a similar tree using the codes in the code dictionary received from the encoding processor, i.e. the RIP processor 2050a. Aside from the imager controller processor 2100a, the decoding processor could serve as a printer controller (not shown) or be some other type of decoding device. The decoding processor performs the reciprocal of the encoding process to decode the encoded sequence characters "102abc100102001102001020110711103". It should be understood that the encoded data could if desired be transmitted to the decoding device via a direct communications link, a local network, a public network such as the Internet, or some other type of network. Further, such communications may be by wire communications or wireless communications. It will also be recognized by those skilled in the art that, while the invention has been described above in terms of one or more preferred embodiments, it is not limited thereto.

[0065] Various features and aspects of the above described invention may be used individually or jointly. Furthermore, although the invention has been described in the context of its implementation in a particular environment and for particular purposes, e.g. imaging, those skilled in the art will recognize that its usefulness is not limited thereto and that the present invention may be beneficially utilized in any number of environments and implementations. Accordingly, the claims set forth below should be construed in view of the full breadth and spirit of the invention as disclosed herein.

3062743.1